

統計的係り受け解析と制約文法の融合による日本語文解析

東 藍, Eric Nichols, 森本 芳弘, 松本 裕治

奈良先端科学技術大学院大学 情報科学研究科

{ai-a,eric-n,yoshi-mo,matsu}@is.aist-nara.ac.jp

統計的統語解析の研究が進み、現実の文に対しても高い精度での解析が可能になってきている。一方、HPSG や TAG 等の豊かな語彙情報をもつ文法の実装も進んでいる。意味処理等のより深い言語処理を行うには、後者のような文法知識が必要であるが、文法制約だけを記述した文法では、一つの文に対して多くの解析結果の候補を出力してしまうこと、また、他方で、僅かな文法制約違反についても厳しく反応して、解析結果を出力できないことが問題となる。本稿では、後者の問題に対して、制約違反を許容した途中解析結果を解析過程で蓄積し、他に好ましい解が存在しない場合にのみ、制約を違反する構成素を解析システムに戻すメタプロセスを導入することで解決を図った。しかし、こうすることで余分な曖昧性を増やしてしまい、解析過程が爆発してしまう可能性がある。この問題については、統計的係り受け解析を制御情報として用い、制約文法の解析を実装することにより、曖昧性の絞り込みを行った。また、係り受け解析結果の利用によって、句構造解析の探索空間の制御だけでなく、日本語で生じるかき混ぜ構文の問題を自然に扱える方法を実現した。

キーワード: 統語解析, 主辞駆動句構造文法, 統計的係り受け解析, 統計情報と制約情報の融合, 頑健な言語解析

Integration of Statistical Dependency Parsing and Constraint based Grammar for Japanese Sentence Analysis

Ai Azuma, Eric Nichols, Yoshihiro Morimoto, Yuji Matsumoto

Graduate School of Information Science, Nara Institute of Science and Technology

{ai-a,eric-n,yoshi-mo,matsu}@is.aist-nara.ac.jp

As advances are made in probabilistic parsing, it is becoming possible to parse even unprocessed text with high accuracy. In comparison, implementation of grammars with rich, lexical data is also progressing. In order to carry out deeper language processing, such as semantic interpretation, grammatical knowledge like that found in HPSG and TAG is necessary. However, when using systems that only make use of grammatical restrictions, many different parses can end up as the output for a single sentence, or due to violation of these rules, a parse may not be successfully generated at all. In this proposal, we offer a solution to the latter problem by permitting and storing parses that violate grammatical restraints, and only in the event that absolutely no error-free parses are generated, a meta-process is used to correct these malformed trees and return them to the parser so that a correct parse can be determined. However, in doing so, the level of ambiguity is increased, and the number of parses generated can explode. In response to this problem, we use dependency information to control how and when trees are produced and in this manner reduce the amount of ambiguity in the output. In addition, by using dependency information, not only are we able to reduce the number of trees generated for a given sentence, but it also becomes possible to handle scrambled sentences in a natural way.

Keywords : Parsing, Head-driven Phrase Structure Grammar, Statistical Dependency Parsing, Integration of Statistic and Constraint Information, Robust Language Processing

1 はじめに

近年、統計的統語解析の性能が向上し、英語の句構造解析や日本語係り受け解析において高い精度を達成している [1][8]。しかし、これらのシステムの出力は通常、

統語解析木であり、より深い言語処理のためには、さらに詳細な文法知識が必要である。一方で、HPSG[10]のような語彙に豊かな情報をもつ文法では、詳細な文法制約や意味解釈を記述することができるが、解析時に生じる膨大な曖昧性の問題や、文法制約に違反する非文法的

文に対する頑健性の欠如が問題になる。

本稿では、統計的日本語句構造解析と制約に基づく文法である日本語 HPSG を融合した言語解析の枠組みを提案する。統計情報と制約情報を融合した言語処理の研究は新規という訳ではなく、本研究は以前の我々の研究 [6] の発展形である。それ以前にも、Schabes[11] による Lexicalized TAG への統計情報の導入や、Kanayamaら [7] による HPSG による解析の曖昧性の絞込みに統計情報を用いる提案がある。

我々の提案は、別々に構築された統計的係り受け解析システムと HPSG 文法とのモジュール性の高い融合である。両者は独立して構築され、特に HPSG は、現実的な文や話し言葉などにも対象を拡げるよう、文法的制約違反をもつ文に対しても制約の緩和を行うことを考えている。これにより、頑健性の問題は対応できるものの、曖昧性の問題がさらに深刻なものとなる。曖昧性と頑健性の問題に対処するため、統計的言語解析システムが HPSG に基づく統語解析のための制御情報を提供する形で解析を行う。ここでの融合を通じて、次のような問題に対処することを考えている。

1. 係り受けシステムの出力である単語依存構造木を句構造解析の制約として用いる方法
2. 日本語のかき混ぜ構文 (scrambling) の取り扱い
3. HPSG における制約違反を起こす非文法的な構造に対する制約緩和処理、および、処理負担の少ない実装

まず、1., 2. について考える。図 1 に句構造木の例を示す。「そびれる」は VP 補語タイプの動詞であり、「直す」を V 補語タイプ、すなわし、語彙複合動詞を構成する動詞である (橋本 [5] を参照のこと)。我々の日本語 HPSG では、郡司 [4] の提案による隣接素性 (Adjacent feature) を用い、図 1 のように、前者が主語を欠く動詞句を隣接要素として要求し、後者が単語としての動詞を要求するものとして記述している。ここで注意してほしいのは、日本語ではこれら両文の格要素の語順を変えるかき混ぜが適用可能で、「日記を健が読みそびれた」「日記を健が読み直した」も正しい文となることである。我々の日本語 HPSG を通常の句構造に基づく統語解析アルゴリズムに適用するだけでは、前者の文をこの語順で正しく解析することができない。

図 2 に南瓜による係り受け解析木の例を示す。我々は、このような出力を HPSG に基づく句構造解析の制御情報として用いる*。日本語の語順の自由度は主として共通の文節に係る兄弟文節間で起こり、係り受け関係にある文節間では起らないことに注意されたい。本稿では、

*現在は、南瓜が出力する第一位の解のみを使うが、曖昧性を含む解析結果を利用することも考えている。

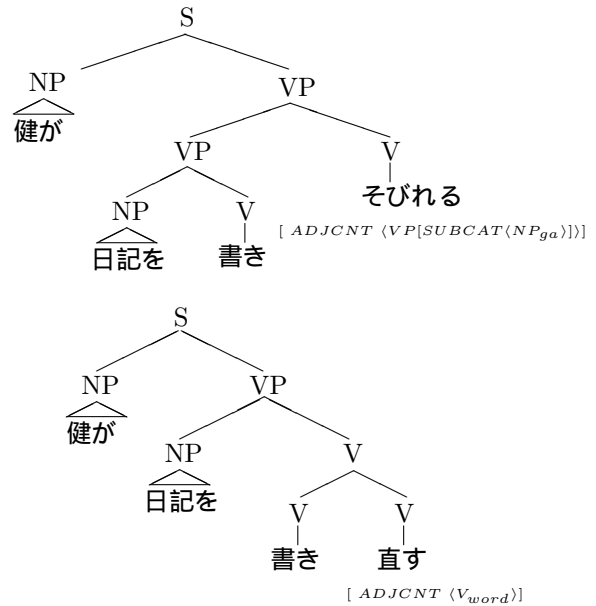


図 1: 複合動詞の解析例

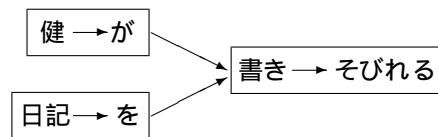


図 2: 文節係り受け解析例

係り受け関係だけを語順の制約として使い、さらに、次のような制約を置くことによって、句構造解析における無駄な解析を抑制する。

- 文節内の係り受けは、文節内に限定される。
- 文節間の係り受けは、係り受け関係のある文節間に限られる。さらに、
 - 係り側は、完成した文節でなければならない。
 - 受け側は、文節内の任意の構成要素が係り受けの対象になる。

この制約にしたがって解析を行うことにより、上記の 2 例の文およびかき混ぜ結果の文に対し、図 1 に示す正しい句構造を得ることが可能になる。

本研究では、次節で示すように、統計的言語解析部と制約に基づく文法解析部を別モジュールとして分離して実装し、緩い形で融合している。その際に、制約に基づく文法処理では、文法制約に違反する部分解析結果を蓄積する別モジュールを追加し、上記 3. の問題に対処している。次々節以降で、それぞれのモジュールについて説明する。

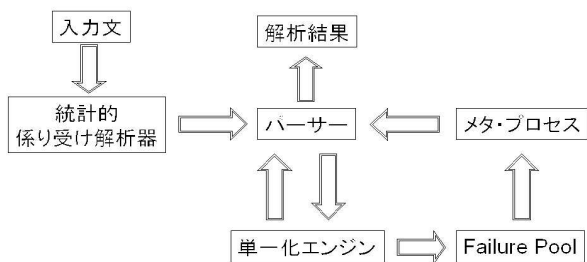


図 3: システムの概観

2 システムの概要

本稿で提案するシステムについてその概要を述べる。まず、概観を図 3 に示す。このシステムでは、まず入力文に対して統計的係り受け解析を行なう。

次に統語解析部がこの入力文の係り受け情報を元にして入力文の統語解析を行なう。通常の構文解析では統語解析の入力は線形の文であるが、これを係り受け木とすることにより解析の曖昧性を軽減しようとするのが本稿で提案するシステムにおける特徴の一つである。この係り受け木を元にした統語解析部については (ref 挿入) 節でその詳細を説明する。

単一化エンジンは統語解析部から渡された句と文法制約を元に単一化を行ない、その文法制約の充足・違反を判定するモジュールである。

Failure Pool は単一化エンジンにおいて失敗が起きた際に、その内容を記録する場所である。

メタ・プロセスは統語解析部で解析が続行できなくなった時に起動されるモジュールであり、Failure Pool 内の情報に基づいて、制約違反の緩和処理を行った結果をパーサーに返す。

Failure Pool 及びメタ・プロセスは、不適格文処理のための付加的な部分であり、入力文が文法的に不適格でない限り、これらが利用されることはない。

3 パーサー

3.1 導入

我々は、かき混ぜ文、つまり不連続構成素という言葉現象を含む文を取り扱えるより効率的な HPSG パーサーを開発する目的では、文法性を HPSG で解釈するための徹底的な構文解析の方法と文の中で数多くの不連続構成から成り立った木構造を保存するデータ構造の必要性が分かる。本稿では、変更した C K Y アルゴリズムと依存関係の情報をともに用いることにより、上記の両方の問題を同時に解決する手法を提案する。この 2 つ

の異なったテクノロジーの役割を具体的に説明をする。

C K Y アルゴリズム [9] とは、動的計画法に基づく文脈自由文法の解析アルゴリズムの一つであり、2 分木構造の文法規則のみを対象とするチャートパーシングと同等である。途中で解釈した木構造を保存するためにチャートと呼ばれる、サイズが入力文の長さ + 1 の 2 次元アレイを用いる。チャートの対角線に文法の規則が入り、チャートの (0,1) から (n,n-1) までかかる対角線の右側の隣にある斜線 (第二斜線と呼ぶが) を入力文で対応する言葉と品詞で初期化される。チャートのエントリーの値は次に式で計算される。

$$\text{chart}(i, j) = \cup(\text{chart}(i, k) * \text{chart}(k, j)), i < k < j \quad (1)$$

$\alpha * \beta$ という演算子は文法規則の適用の成功を示す。充足の条件は、 α が文法規則の左の子構成素に当たり、 β が文法規則の右の子構成素の場合である。構文解析の成功は、長さ n の入力文に対し、 $\text{chart}(0, n)$ が文法の開始記号の S を含むことで定義する。このようなアルゴリズムが $O(n^3)$ 桁で実行されるのは明白であろう。

パーサーの効率を高める方法の提案は多い。例えば、Earley アルゴリズム [2] では文法の複雑さによって異なるが、 $O(n), O(n^2)$ で実行できる場合もある。また Valient [12] は大きさが n のマトリックスを 2 つ掛算する時間で走る手法を提案した。我々が提案する手法では、 n の値の最小化で効率上がるわけである。このために、依存関係という別の言語情報源を用いる。依存関係の情報を用いると、C Y K パーサーにとっての単語の接続性が係り受け構造で定義されるため、従来の定義とは異なるという問題が発生する。これを避けるために役立つデータ構造を考える。

我々の開発したパーサーの実装では、入力文の依存構造と品詞の情報を受けるために係り受け解析システム南瓜 [8] を利用する。南瓜は、Support Vector Machines に基づく学習モデルを用いる日本語係り受け解析器である。係り受け木のノードでカーバーされた入力文の部分、あるいは親子ノードの接続関係にのみ限定された木構造の生成だけを許可するためにこの情報を利用する。さらに、すべての子ノードが解析されるまでノードの解析を起こさないという制限も与える。例えば、図 4 は「直美が健が書いた本を読みそびれた。」の係り受け木である。ルートノードの「読みそびれた」を解析する前に、「直美が」と「本を」の 2 つの子ノードを解析しなければならない。チャートエントリーの値を計算する時に、選択された構成素が依存構造中どのノードにしか関係しないかを調べ、上記の制限にしたがってエントリーを書き込むかを判断する。たとえば、「健が」のノード

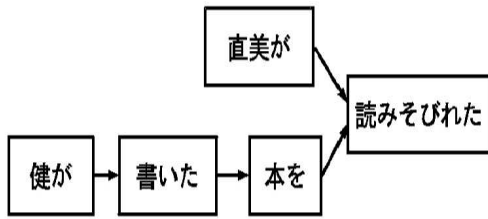


図 4: 「直美が健が書いた本を読みそびれた。」の係り受け木

からの構成素を「書いた」にしかかけないようにする。それで、 $n * n$ のサイズのチャートを構成するためのコストが依存構造のノードに対応する小さいチャンク合計となるようにすることが可能である。

ある依存構造のノードから子ノードの数とノードに含む単語の総数を z とすれば、解析のコストは $\sum(z^3)$ になる。実行時間の計算の例を挙げよう。 z の値が同じ 2 つの子ノードがある依存構造に対応する入力文があるとして。その場合、実行時間は、 $(n/2)^3 + (n/2)^3 = (n^3)/4$ と計算できる。同様にサイズが同じ 3 つの子ノードがある依存構造に対する計算時間は、 $(n^3)/9$ になる。実際には、 z の最大のものの値が最も影響を与えるために、われわれのアルゴリズムは $O(z^3)$ だと言える。

依存構造のノード数は高々入力文の語数に過ぎないため、最悪のケースでも CYK アルゴリズムと同じオーダーであることが証明できる。しかし、実際には z が n より値が小さいことがほとんどであるために我々の提案するアルゴリズムのほうが効率が低い。かき混ぜ文を取り扱うために、親ノードに子ノードがかかる順番は可能な全ての場合を試さねばならない場合には、すべての途中結果に対応する木構造を生成してしまう。親のノードが最右の位置なので、アルゴリズムの実行時間は、 $(c!)(z^3)$ に増大する。ここに、 c は子供のノードの数であり、すべての子ノードのかかる順番を考慮する意味になっている。現在、HPSG における処理の制限を生かすことにより、かき混ぜ文を処理する際にすべての組み合わせを考慮するコストを減らすための研究を行なっている。

3.2 パーサーの実装

この節では、パーサーの実装を説明する。まず、パーサーのデータ構造から始める。パーサーは入力として文と依存構造を受け、この依存構造のノードを拡張して基盤的なデータ構造にする。各ノードをアレイに保存し、ノードの ID 番号をアレイの位置として定義する。そして各ノードにかかる子ノードの ID 番号をリストに

```

PROCESS_NODE (CURR_NODE)
  CURR_NODE.PROCESS = TRUE
  NEW_TREE_IDS = CALC_TREE_IDS (CURR_NODE.ID, CURR_NODE.CHILDREN)
  FOR EACH TREE_ID IN NEW_TREE_IDS
    GENERATE_TREES (TREE_ID)
  IF CURR_NODE.TREES CONTAINS TREE THAT COVERS CURR_NODE
    AND CURR_NODE.CHILDREN
    CURR_NODE.SATISFIED = TRUE
  ELSE
    CURR_NODE.TREES += ERROR_RECOVERY (CURR_NODE)
  END

CALC_TREE_IDS (ID, CHILDREN_IDS[ ])
  RETURN LIST OF ALL PERMUTATIONS OF CHILDREN_IDS FOLLOWED BY ID
END

GENERATE_TREES (TREE_ID)
  NODE_IDS = TOKENIZE (TREE_ID)
  FOR EACH NODE_ID IN NODE_IDS
    CURR_NODE = NODES[NODE_ID]
    IF NOT CURR_NODE.PROCESSED
      PROCESS_NODE (CURR_NODE)
      NEW_TREES[ ] = CKY_PARSE (NODE_IDS)
      PARENT.TREES[ ] += UNIFY_VERIFY (NEW_TREES)
    END
  END

```

表 1: パーサーの pseudocode

して持つ。また、ノードでカバーされる入力文の部分と品詞情報もノードのデータ構造に加える。最後にノードに対応する木構造を保存するためのアレイも含む。

パーサーが実行される時、最初に PROCESS_NODE という関数を依存構造のノードで呼び出す。PARSE_NODE は管理の役割を果たし、ノードの処理に必要なタスク行なう。実際に木構造を生成する処理を呼んだり、解析が終わったか充足したかのようなノード処理の現状を記録したりする役割がある。PROCESS_NODE が新しいノードで呼ばれる時に、無限ループが起きないように NODE.PROCESSED を真にして、ノードの処理が始まったという意味の旗を立てる。そして、CALC_TREE_IDS を呼び出し、生成する木構造の ID 番号のリストを作成する。

CALC_TREE_IDS は子供ノードの ID の順列をすべて文字列として生成し、各ストリングの後ろに親ノードを付けてリストの形で PROCESS_NODE に渡す。例えば、CALC_TREE_IDS の入力が (1, [2, 3]) の場合には、出力が ("2.3.1", "3.2.1") となる。ノード ID の間にドットがあるのは、ノード ID に区切る可能を考慮するためである。

我々のパーサーでかき混ぜ文を取り扱うために、ある依存構造のノードの子供ですべての純列に当たる木構造を生成することが必須である。HPSG 文法では、構成素のバインディング順が重要である。例えば「読む」という動詞は SUBCAT 素性でガ格の名詞とヲ格の名詞を要求する。「読む」は「そびれる」という補助動詞にかかることがある。「そびれる」は SUBCAT でガ格の名詞だけを受けるために「読む」と単一化される際に「読む」の SUBCAT にヲ格の名詞が残っていることを

許さない。したがって、「読む」と「そびれる」を単一化する前に、ヲ格の持つ名詞を「読む」にかける必要がある。逆に、「読む」と「そびれる」のガ格名詞が同じ項を指しているため、ガ格名詞を先にかけようとするれば、「そびれる」の方がガ格名詞の不足のために失敗する。CALC_TREE_IDS を用いると、適切な木構造の生成が確認できる。

CALC_TREE_IDS が終わると、PROCESS_NODE が GENERATE_TREES という関数を呼び出す。GENERATE_TREES は木構造 ID のリストを受け、木構造 ID に並ぶ順番で示されるノードの木構造を生成する。CYK アルゴリズムで木構造を生成する前に、各ノードで PROCESS_NODE を呼び出す。これにより、あるノードの子供が帰納的に親の前に処理されて木構造が生成される。CKY_PARSE で生成される木構造の文法性を解釈するために UNIFY_VERIFY を呼ぶ。

UNIFY_VERIFY は、木構造のリストを独立したモジュールとして実装された単一化エンジンに渡す。この単一化エンジンが HPSG の文法規則を適用し、木構造の文法性を判断する。非文法的な木構造を認識すると、Failure Pool にその結果を格納する。成功した木構造のリストを GENERATE_TREES に戻し、GENERATE_TREES がそのリストを親のノードの木構造のリストに足す。単一化エンジンと Failure Pool の詳細は次節で説明する。

最後に GENERATE_TREES の実行が終り、PROCESS_NODE に戻る。そして、PROCESS_NODE がノードの木構造のリストを通り、ノードと子供をすべて含むパースを検索する。適切なパースがある場合には、NODE.SATISFIED を真にして解析を成功と判断する。検索が失敗する場合には、ERROR_RECOVERY を呼び、単一化エンジンに Failure Pool にある非文法的な木構造から最も制約緩和が容易な木構造の選択を依頼する。適切な木構造を Failure Pool から受け取ると、処理が続ける。

4 文法的不適格文処理

通常の構文解析システムは、文法的に適格な文を対象としている。しかしながら、実際の言語使用においては文法的に不適格な文も多く、これらの文は従来のシステムにおいては文法的に不適格であるとされ、システムは解析に失敗する。

このような文に対する 1 つのアプローチとして、あらかじめ分かっている不適格性に対して、明示的な規則を書き下す方法が考えられる。しかし、この方法では通常の適格文に対する解析において冗長な曖昧性を生じ

させる恐れがある。

そのため本稿で提案するシステムでは、あくまで文法的不適格文処理を解析失敗時の付加的な処理とすることにより、通常の適格文の解析に冗長な曖昧性や、余分な処理の負荷を生じさせることなく不適格文処理を行なうことを意図する。

我々はこの目的のために、ある素性の単一化に失敗しても単一化演算を強制するよう単一化エンジンを改良し、さらに単一化失敗の内容を記録する Failure Pool、そして解析失敗を修正するメタ・プロセスという 2 つの部分システムを組み込む。

5 単一化エンジン

単一化エンジンは、パーサーとは独立したモジュールであり、句の素性構造に対する単一化演算を行ない、その成否によって文法制約を満たしているかどうかを判別する。

通常の単一化演算では、演算が成功した場合にはその結果を返し、また何らかの矛盾によって失敗した場合には、パーサーには何も返さない。

しかし、本稿のシステムにおいては、単一化に失敗した、つまり文法制約に違反する部分解析結果を救うという意図もあるため、矛盾の具体的内容を記録する必要がある。そこで、本システムにおける単一化エンジンは対象とする語句の素性構造間に矛盾があった場合には、パーサーに失敗であるという結果を Failure Pool と呼ばれる場所にその矛盾の内容と共に記録しておく。

具体的に、次のような素性構造間の単一化を考えてみる。

$$\left[\text{HEAD } \textit{verb} \right. \\ \left. \text{VAL } \left[\text{SUBCAT } \left[\text{HEAD } \left[\textit{noun} \right. \right. \right. \right. \\ \left. \left. \left. \left. \text{CASE } \textit{ga} \right] \right] \right] \right] \right] \quad (2)$$

$$\left[\text{HEAD } \left[\textit{noun} \right. \right. \\ \left. \left. \text{CASE } \textit{wo} \right] \right] \quad (3)$$

この 2 つの素性構造間の文法制約として、(2) の SUBCAT の値と (3) の値が一致していなければならないという場合を考える。この場合、CASE の値に矛盾を引き起こし、単一化エンジンは素性構造の単一化に失敗する。しかし、矛盾する部分を記録したまま単一化処理を進め、矛盾を含む単一化結果を Failure Pool に格納する。この例の場合は、CASE の値に *ga* と *wo* の矛盾が起きたことを記録した以下のような素性構造が Failure

Pool に書き込まれる .

$$\left[\begin{array}{l} \text{HEAD } verb \\ \text{VAL } \left[\begin{array}{l} \text{SUBCAT } \left[\begin{array}{l} \text{HEAD } \left[\begin{array}{l} noun \\ \text{CASE } \perp(wo, ga) \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right] \end{array} \right] \quad (4)$$

ここで, \perp は矛盾を表す記号である .

これと同時に, 各々の素性構造の単一化の失敗に対して制約の違反の度合を示すコストを計算し, その値も Failure Pool に格納しておく . コストの計算方法としては, 例えば以下のように計算できると考える .

$$\sum_{f \in F} w_f \quad (5)$$

ここで, w_f は素性 f における矛盾に対するコスト, F は素性構造中で矛盾している素性の集合である .

6 メタ・プロセス

入力文が文法的に不適格であった場合には, パーサーにおいてすべての部分解析結果が失敗する . この時に初めてメタ・プロセスが起動される . メタ・プロセスは Failure Pool が保持するすべての部分解析結果の中から解析の続行に最も有効と考えられる (制約の違反の度合いが最も軽微な) 部分解析結果を選択し, それを修正する . 問題は, メタ・プロセスがこの部分解析結果をどのように修正するかである . 以下ではその概要について例を上げつつ述べる .

6.1 部分解析結果の選択

メタプロセスは, Failure Pool に記録されているコスト値に基づいて部分解析結果を選択する .

具体的に「健が本読む」という文について考える . この入力文に対してパーサーは「健が + 読む」と「本 + 読む」の 2 つの部分解析結果が失敗するために, 解析を続行できない . †ここでメタ・プロセスが起動され, プロセスは Failure Pool の中から解析失敗の原因の軽いものを選択する . 各部分結果のコストは, 次のような原因により計算されている .

「健が + 読む」の部分解析結果の失敗の原因は「読む」が SUBCAT の値としてヲ格の名詞を要求しているのに対して「健が」の格がガ格であることである . 従って, この場合「読む」の SUBCAT の CASE 属性に矛盾が生じていることになる . 一方「本 + 読む」は「本」の CASE 属性の値が定まっていないため SUBCAT の

†本稿のシステムでは, 動詞の SUBCAT における格の順序を固定した文法を想定していることに注意されたい

CASE 属性には矛盾は生じない . しかし「本」が格マーキングされていないという制約違反に対応する素性に矛盾が生じる . ここで, 格マーキングがされていないという矛盾のコストが, CASE 属性における矛盾のコストよりも低いとすると「本 + 読む」の修正が優先されることになる .

6.2 部分解析結果の修正

コスト付けによって, 最もコストが低いとされた部分解析結果に対して, プロセスは修正を行なう .

例として (4) における矛盾「 $\perp(wo, ga)$ 」を考えてみる . 最も単純には素性構造中の矛盾した値である「 $\perp(wo, ga)$ 」に対して, いずれかの値を取り出してその値で直接置き換えるという修正方法が考えられる .

しかし, 素性構造中の矛盾した素性と値のみを参照するだけではどのような種類の矛盾がおきているのか知るには不十分なことが多い . 上の例では「 $\perp(wo, ga)$ 」を参照しただけではどちらが名詞句由来の値でどちらが動詞句由来の値が分からない . これはつまり「 $\perp(wo, ga)$ 」を「 ga 」と「 wo 」のどちらに修正するのか, つまり動詞句の下位範疇化構造を修正するのか, 名詞句の格マーキングを修正するのか判別できないことを意味する . このままでは修正に曖昧性を残すこととなり, 可能な修正が膨大になる懸念がある . このような問題点から, 矛盾を生成した文法制約と元の語句も参照して総合的に修正方法を決定する必要があると考える . あるいは, Fouvry[3] が提案するように, 各素性の値に重みをつけておき, その重みに基づいて取るべき値を決定することも考えられる .

また, 矛盾した素性の値を直接修正する以外に, 別の素性の値を修正する可能性も考えられる . これは「健 + 食べる」の例を考えてみると, 矛盾した値を直接変更する方法では「健(ヲ)食べる」という修正が得られるが, 同時に「健(ガ)(ヲ)食べる」という修正も十分可能であることから分かる . このような修正を行なおうとすると, 矛盾した値を直接変更するだけの修正ではもはや不十分であることは明らかである .

そこで, 2 つの語句が制約に違反した時に, 一方の素性構造の値を他方が要求する値に強制する (この操作は上記の直接書き換えに対応する) だけではなく, 一方の語句の値 (これは矛盾した素性の値とは限らない) を書き換えつつそれに対応するかたちで他方の値を書き換える (あるいは制約を緩和する) といった操作も考えてみる .

上の例では「健 + 食べる」に「健(ガ)食べる」という強制 (「健」の格をガに強制する, あるいは「食べ

る」の下位範疇化構造を〈ガ, ガ〉とする)を施す修正の他に, 不指定 *case* である「健」CASE 素性を特定の格 *ga, wo, ni, ...* に指定しつつ別の修正を施す, という操作も考慮してみる。「健 + 食べる」に対してはまず「健」を「健(ガ)」と特定した後, さらに「食べる」の SUBCAT のヲ格がすでに埋まっているものに変更すると「健(ガ) (ヲ) 食べる」のような形に修正できる。

このように, 2つの語句の素性構造間に矛盾が発生した時に, 双方の(矛盾を来した素性以外の)素性に変更を加えることにより, より高度な修正が実現されうると考える。

具体的に, 以下のような変更と強制を考える。

- 特定されていない格を特定する
- 特定の格を別の格に変更する
- 空の SUBCAT を空でないものに変更する

各々の修正にはコストを設定しておき, あるコスト以上の修正は行なわないとして修正の探索空間を限定する。

動詞の下位範疇化において, 格一致に対する制約違反が起きた時を例にとると, 次のような修正が得られる。

- 名詞の格がマーキングされている時
 1. 動詞の SUBCAT の変更を行なう ($cost = c_1$)
 2. 名詞の格を動詞の要求格に強制する ($cost = c_2$)
- 名詞の格がマーキングされていない時
 1. 名詞の格マーキング素性を強制する ($cost = c_3$)
 2. 名詞の格の特定と動詞の SUBCAT の変更を同時に行なう ($cost = c_1 + c_3$)

ここでは, コストの大小として仮に $c_3 < c_1 < c_2$ としておく。またコストの積算は簡単のために和を取るものとしておく。

では, 具体的にどのように部分解析結果が修正されるかを, 以下の助詞が欠落した文章を対象としてみる。

健ご飯食べる (6)

この入力文に対する解析の結果「健 + 食べる」と「ご飯 + 食べる」の2つの部分解析結果が矛盾を含んだ状態となって Failure Pool に記録され, 解析は一時停止する。

まず, メタプロセスは修復すべき失敗を Failure Pool から取り出す。この場合は, 上記2つのコストに差がないため, 両方が修正の対象となる。次にメタプロセスは, 個々の失敗に対して可能な修復方法を探索する。「健 + 食べる」に対しては, 健の格をヲ格に特定する方法 ($cost = c_3$) と, 食べるの SUBCAT を変更した

のち健の格をガ格に特定する方法 ($cost = c_1 + c_3$) などが考えられるが, ヲ格を特定する方が低コストであるため「健(ヲ) + 食べる」と修復する方が優先される。同様に「ご飯 + 食べる」に対しても「ご飯(ヲ) + 食べる」という修正 ($cost = c_3$) が行なわれる。この2つはこの段階では等コストであるため, パーサーにこれら2つの修正された部分解析結果を渡し, 解析を続行させる。

しかし, パーサーはすぐに「ご飯 + 健(ヲ) 食べる」と「健 + ご飯(ヲ) 食べる」という新たな矛盾に直面し, 再びメタプロセスを呼び出す。上記2つに対してはともにガ格に特定する修復が上げられる。従って, 両方ともにコスト $2c_3$ で「ご飯(ガ) + 健(ヲ) 食べる」と「健(ガ) + ご飯(ヲ) 食べる」に修正される。しかし, 「ご飯(ガ) 健(ヲ) 食べる」は単一化においてさらに意味的制約での矛盾を来すと考えるので, 最終的に「健(ガ) ご飯(ヲ) 食べる」の解析がコスト $2c_3$ で生成されることになる。

6.3 今後の課題

今後の目標として, まずは実際のコーパスで稼働させた際にどのような失敗が発生するのかを捉える必要がある。このために, 本システムを Failure Pool 及びメタ・プロセスを切り離れた状態で実際のコーパス, 特に文法的に不適格な文の多い談話コーパスのようなものに対して稼働させて, 単一化の失敗の結果を蓄積しなければならない。その上で各素性に対する失敗の頻度から各素性のコストを計算していくことになる。また, 値を直接書き換えるだけでは対応できない高度な修正に対しては, その枠組をさらに精緻化しなければならない。

7 おわりに

本稿では係り受け解析木を入力として用いることにより, CYK パーサーの効率化を図る手法を示した。同時に, 不連続構成素を持つ文に対して本手法がどのように適用されるのかも示した。しかしながら, 本手法では1ノードが持つ子供のノードの最大数の階乗に比例するため, 本手法はこの最大数が小さい係り受け解析木に対してのみ有効である。現在我々は, かき混ぜのある文を考慮した時に, 解析の数を減らす方法を研究している。

また, 我々は文法的に不適格な文に対しても頑健に解析する手法を提案した。パーサーが解析に失敗したときのみ不適格文の修正を行なうことにより, 通常適格文の解析において冗長な曖昧性を生じさせることなく頑健なシステムを構築できることを示した。今後は,

部分解析結果に対するコスト付けをどのように定式化するのか、また、より高度な修正を行なうための枠組をどう構築していくかが今後の課題となる。

謝辞

日本語 HPSG 勉強会メンバーの皆さんの日頃のご意見に感謝します。特に、大阪学院大学大谷氏、神戸松蔭女子大大学院橋本力氏、NTT-CS 研の Francis Bond 氏には、この勉強会や日頃の議論において多くの示唆をいただきました。ここに感謝します。

参考文献

- [1] Charniak E (2000) “A Maximum-Entropy-Inspired Parser.” In *Proceedings of 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pp.132-139.
- [2] Earley J (1970) “An Efficient Context-Free Parsing Algorithm,” *Communications of the ACM*, 13 (2).
- [3] Fouvry F (2003), “Constraint Relaxation with Weighted Feature Structures,” In *Proceedings of 8th International Workshop on Parsing Technologies*, pp.103-114.
- [4] Gunji T (1990) “On Lexicalist Treatment of Japanese Causatives,” In *Studies in Contemporary Phrase Structure Grammar*, Cambridge University Press, pp.119-160.
- [5] 橋本力 (2003) 「日本語 HPSG:統語的複合動詞の統語・意味構造の処理」情報処理学会自然言語処理研究会, 2003-NL-156, pp.31-38.
- [6] 今一修, 松本裕治, 藤尾正和 (1998) 「統計情報と文法制約を統合した統語解析手法」自然言語処理, 5, (3), pp.67-83.
- [7] Kanayama H, Torisawa K, Mitsuishi Y, Tsujii Y (2000) “A Hybrid Japanese Parser with Hand-crafted Grammar and Statistics,” In *Proceedings of the 18th International Conference on Computational Linguistics*, (1), pp.411-417.
- [8] Kudo Y and Matsumoto Y (2002), “Japanese dependency analysis using cascaded chunking,” In *Proceedings of Sixth Conference on Natural Language Learning (CoNLL-2002)*, Taipei, Taiwan, pp.63-69, August-September 2002.
- [9] Jurafsky D and Martin J (2000): *Speech and Language Processing*, Prentice Hall.
- [10] Sag I A, Wasow T, Bender E M(eds.) (2003) *Syntactic Theory: A Formal Introduction*, Csl Lecture Notes, No.152.
- [11] Schabes Y (1992) “Stochastic Lexicalized Tree-Adjoining Grammars,” In *14th International Conference on Computational Linguistics*, (2), pp.425-432.
- [12] Valient L (1975), “General Context Free Recognition in Less Than Cubic Time,” *J. Computer and System Sciences*, 10, 308-315.